DEVELOHMMW'01~ A COMMERCIALLY VIABLE, MODULAR AUTONOMOUS ROBOTIC SYSTEMS FOR CONVERTING ANY VEHICLE TO AUTONOMOUS CONTROL

David W. Parish
Robert D. Grabbe
Omnitech Robotics, Inc.
Englewood, CO. 80110

Dr. Neville 1. Marzwell
Jet l'repulsion Laboratory
California Institute of' Technology
Pasadena , CA. 91109

ABSTRACT

A Modular Autonomous Robotic System (MARS), consisting of a modular autonomous vehicle control system that can bc retrofit on to any vehicle to convert it to autonomous control, and support a modular payload for multiple applications is being developed. The MARS design is scalable, reconfigurable, and cost effective due to the usc of modern open system architecture design methodologies, including serial control bus technology to simplify system wiring and enhance scalability. The design is augmented with modular, object oriented (C++) software implementing a hierarchy of five levels of control including teleoperated, continuous guidepath following, periodic guidepath following, absolute position autonomous navigation and relative position autonomous navigation. The autonomous vehicle control system design uses a stochastic map, and cascaded Kalman filter to fuse numerous position sensor groups, including an inertial sensor suite, a differential GPS sensor, several landmark detection sensors, and a cost effective, random access 3600 scanning laser rangefinder, or LADAR. The LADAR also doubles as a high precision obstacle detection sensor. operational capability of a rapid prototype ATV has been demonstrated including the LADAR, machine vision, and inertial sensor suite based dead-reckoning. The present effort is focused on producing a system that is commercially viable for routine autonomous patrolling of known, semi-structured environments, like environmental monitoring of chemical and petroleum refineries, exterior physical security and surveillance, perimeter patrolling, and intra-facility transport applications.

INTRODUCTION

Numerous autonomous robotic vehicles and control systems have been developed in recent years, by universities, government labs, and commercial companies. Some of these designs have found commercial applications, although broad based commercial application and market acceptance has been illusive for all but the simplest approaches like Automated Guided Vehicles (AGVs). Based on market analysis, applications for autonomous robotic vehicles are apparently plentiful, if a satisfactory system and life-cycle cost effectiveness can bc met. We hypothesize that with a properly modularized, scaleable, and reconfigurable autonomous

vehicle control system architecture, that a commercially viable system can be obtained, providing sufficient cost effectiveness and return on investment to justify substantially increased market acceptance for a variety of applications. This paper will introduce our technical approach for producing such a system, followed by an overview of some of the candidate applications targeted.

TECHNICAL APPROACH

"l-hc fundamental basis of our approach for the development of the "Modular Autonomous Robotic System" or MARS.is based on recognizing that it is possible, and highly desirable, to separate the vehicle being controlled from the autonomous control system itself. This allows the use of any vehicle, whether general purpose or specially built, and by subsequently adding on a set of sensors, actuators, and "black box" electronics and control computers, an intelligent autonomous robotic vehicle can be produced for virtually any application. This approach forces a generalized architecture, and limits the use. of simplifications for some applications. However, it is anticipated that the economics of scale and extra flexibility provided by having a general approach will ultimately be more advantageous. Modularizing an autonomous robotic vehicle control system requires consideration of various aspects, including:

- computing hardware, both centralized and distributed
- actuation hardware
- sensing hardware
- communication hardware and software
- control algorithms and software
- software infrastructure

The scope of these technologies is extensive, and creating an architecture that handles all aspects can seem overwhelming. However recognizing that the architecture will ultimately be embodied as one or more physical instantiations, wc approached the problem by starting with a formal product specification sheet for both a low end indoor AGV-like autonomous robotic vehicle, and a high end outdoor security and surveillance autonomous robotic vehicle. These two designs were then generalized, and the common functions and interfaces defined, to provide a scaleable architecture that could meet both design targets cost effectively. Wc will describe some of the preliminary approaches taken and results of this effort, by describing the hardware architecture first, followed by the software architecture.

MARS Hardware Architecture

Figure 1 shows an overview of our hardware architecture for the MARS development. Onc of the main features of this hardware architecture is the usc of a serial control bus for acquiring sensor information and controlling actuators. This "sensor / actuator bus" will now be introduced.

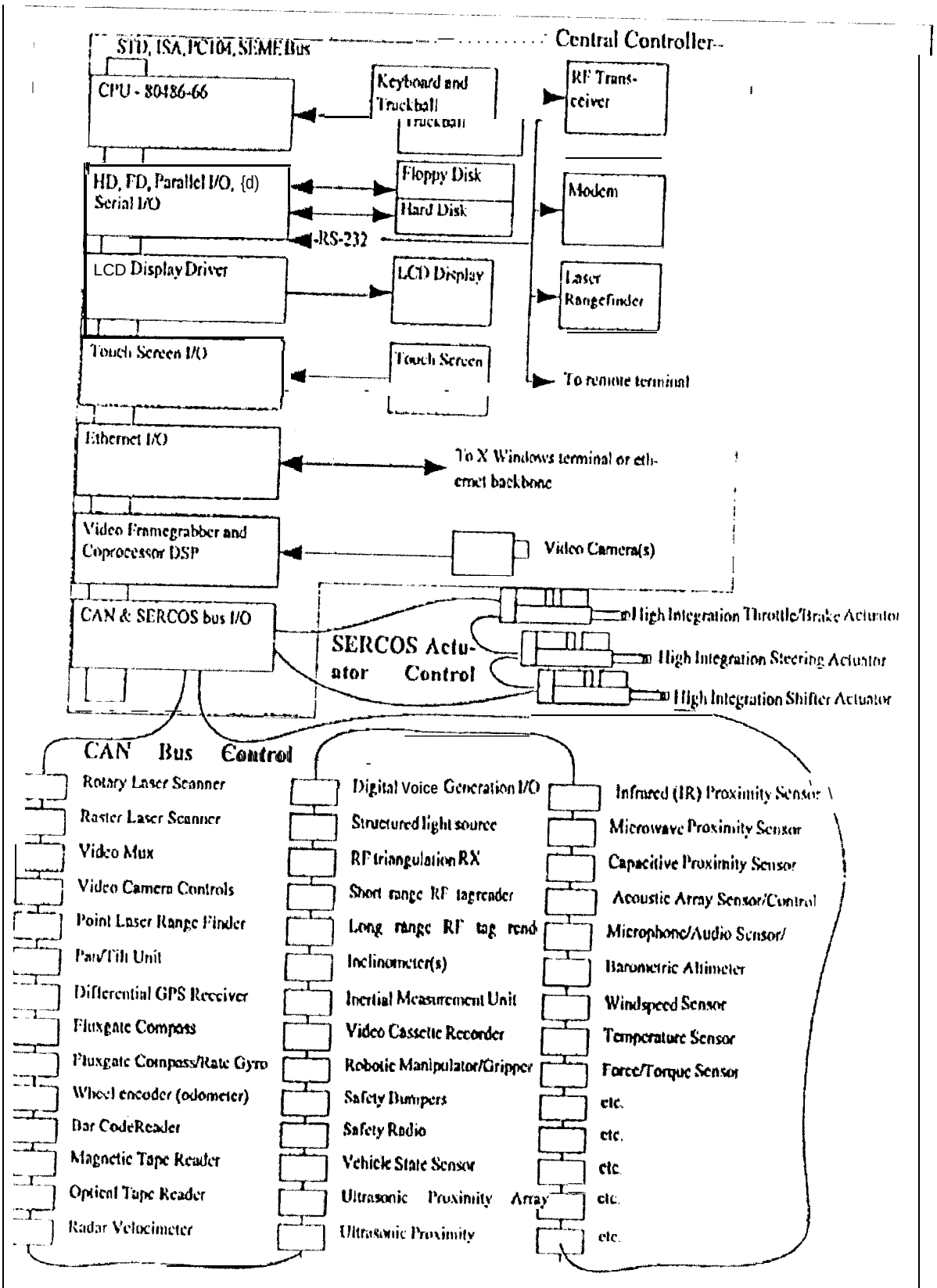

Figure I: Overview of the MARS Hardware Architecture

Serial Control Bus '1'ethnology

The concept of a scaleable controller bus has been developed to increase the level of integration involved in embedded controller applications. Specifically, by converting from a parallel hardwired electrical interconnection approach to a distributed serial bus interconnection approach, significant savings in wiring and other raw materials, and installation costs can be gained.

The serial bus interface approach to embedded controls has been spear-headed by major automotive manufacturers to reduce in-vehicle wiring cost and size for new automobiles, for the control of electric windows, lights, accessories, and similar items. The primary automotive bus standard has been Controller Area Network, or CAN, which has been promoted by Bosch and other major manufacturers.

CAN interface chips are now available from Intel, Motorola, Phillips, Signetics and others, and the application of this approach is accelerating. These hardware components provide the physical and data layer functionality, but lack a standardized application level protocol for interoperable communications. Some work at standardization of the 0S1 ISO level 7 application protocol for the CAN bus has been conducted, and Omnitech Robotics is currently active in this area. Omnitech Robotics has recently completed the development of it's first CAN bus interface product, called CANAMP. CANAMP provides the following features:

- Networkable motion control
- 600 watt brush motor servo amplifier
- 32 bit]DSP motion controller
- Microcontroller with BASIC interpreter
- 1 Mbaud CAN interface (ISO/DIS11898)
- Digital amplifier parameter setting
- 8 analog inputs, 10 bit resolution
- 6 digital inputs, 2 digital outputs
- optional analog tachometer stabilization

A photograph of a CANAMP is shown in Figure 2.

nigh integration Actuators

High integration actuators refers to a design approach where the servo actuator is packaged with the necessary control components into a complete stand-alone unit. Specifically these units typically incorporate a motor, transmission device, feedback element(s), power amplifier, and logic control ler with interface in a single hardware package.

The advantage of using high integration actuators include the ability to completely specify the resulting actuator's performance parameters, reduction in the total system weight, size and volume duc to the elimination or minimalization of ancillary connectors, cables, etc., and the unit is convenient to USC, mount, test, and replace.

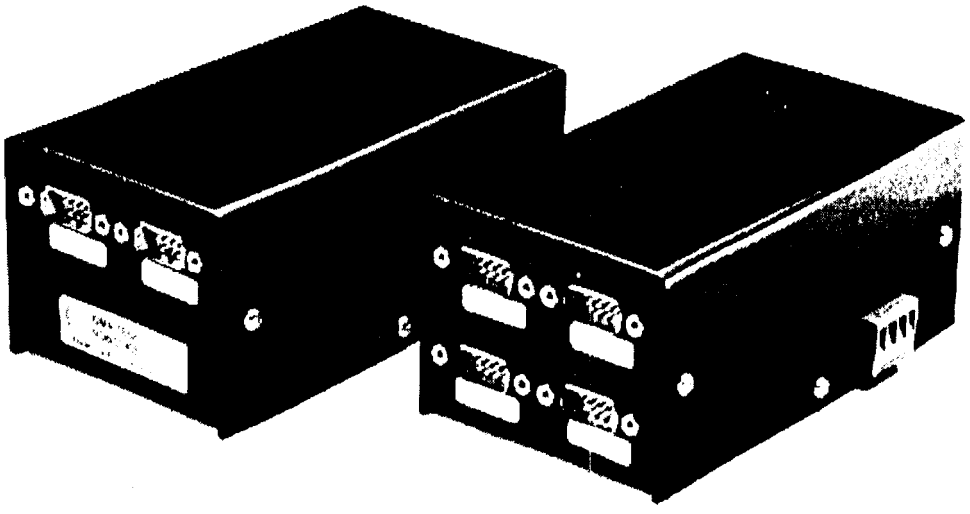

Figure 2: ***Photograph of CANAMP***

The concept of high integration actuators is not new, in fact it has been applied to aerospace type applications for many years, due to these applications' premium on size, weight, volume, and performance. It is new to main-stream automation applications however. Figure 3 illustrates an overview of high integration actuators developed by Omnitech using the CANAMP.

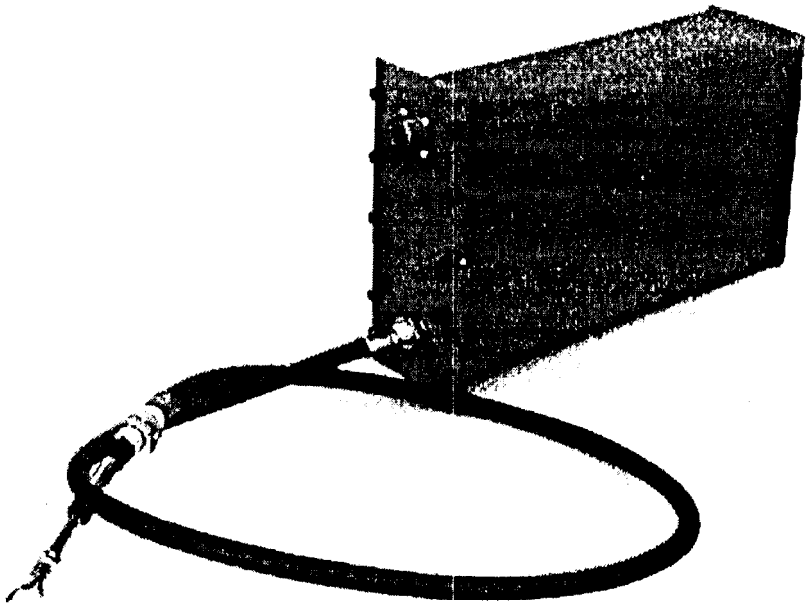

Figure 3:1'11 ***otograph of the High Integration Actuator Developed by Omnitech Robotics, Inc.***

Architecture for Configuration Management

The vision of this work is a development architecture that establishes and supports a managed information repository of modular hardware and software elements that can be installed into Unmanned Ground Vehicle (UGV) systems.

Figure 4 shows an overview of the combination] of tools and methods to accomplish a configuration management architecture. Configuration Management will maintain documented software source code and documented hardware configuration modules available al the design, analysis, and implementation levels of system development. This would reduce the amount of re-engineering and enhance the interoperability of teleoperated / semi-autonomous systems.

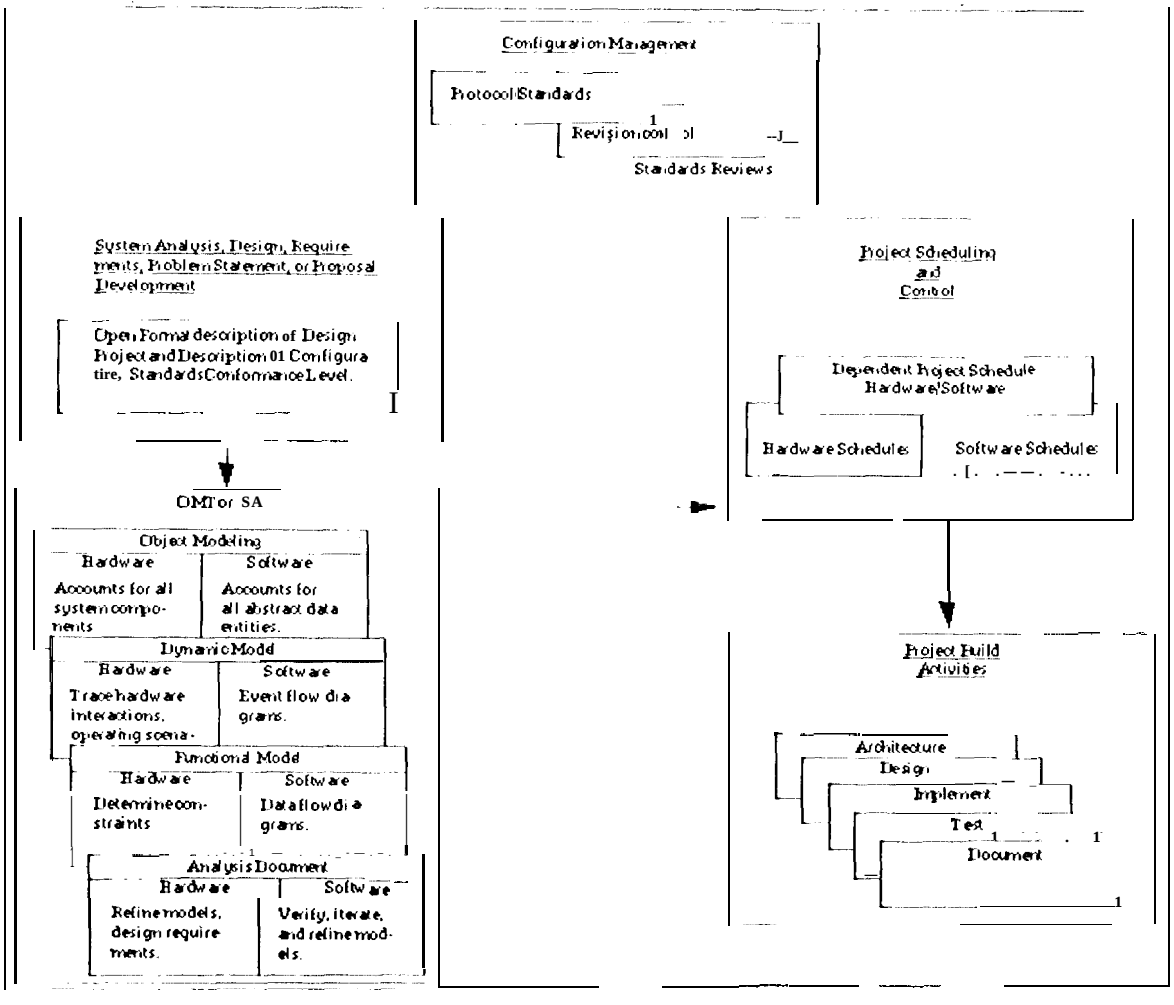

FIGURE 4: An Object Modeling Technique (OMT) emphasizes the naming of all physical objects (hardware) as well as abstract (soft ware) objects allowing for early scheduling of th c development task.

Configuration management will encompass the tasks of handling changes to software/ hardware components that comprise the interface protocols that maintain intra and interoperability of UGVs developed under a joint architecture. This includes methods for evaluating changes, tracking changes, and keeping copies of the architecture that existed at various points in time. The complexity of this task requires a systematic approach that is embodied in our outline of a configuration management architecture.

Initially, our configuration Management architecture will specify and maintain a basis set of protocol standards, software and hardware, available for the usc of achieving downwardly compatible intra and interoperable UGV systems.

The support for the managed information repository will consist of automated project management tools, revision control tools, and real-time structured analysis tools such as the Object Modeling Technique (OMT) combined to manage a joint UGV configuration architecture.

Managed hardware elements will consist of controller networking such as the Controller Area Network (CAN) nodes and high integration actuators to be used on the vehicles, radio equipment, and operator control unit (OCU) equipment and standards.

Software components will consist of modularized units for communication protocols, software-hardware interface, data elements and structures, and message formats.

Documentation of the hardware / software elements will address the intended USC, applied uses, and reproducible test cases and results. This type of documentation would make the evaluation of hardware / software reuse feasible.

METHODS AN1) TOOLS OVERVIEW

Operating Systems and Programming Software

The platform for system development would be a real-time operating system, using standardized 0S services (like the IEEE defined POSIX 1003.1 and 1003.4 standards) such as the LynxOS to support threading of processes for rate monotonic process organization, and the Ada or C++ programming language.

Structured Analysis

Structured analysis tools decompose a design task into smaller, more manageable subsystems. The Object Modeling Technique (C)MT) developed by Rumbaugh allows for a system decomposition based upon the objects in the system regardless of whether they arc hardware components or software components.

System components from a Configuration Management repository could be evaluated and reused at this early point in the design process. Their applicability to a current system design

would be verified by the documentation giving a module's intended USC, reproducible test cases, and systems currently using the module.

Project Management Tools

Automated time scheduling tools follow the object modeling process by incorporating the hardware / software objects into a scheduling process.

Revision Control Software

To maintain configuration control over systems interfacing, system modules used from a repository arc compared to the originals for changes. If the than.gcs warrant permanent inclusion into the standardized protocols while maintaining downward compatibility with other systems, then they arc added to the standards and given a revision number. This aspect should not be underestimated, in fact Microsoft corporation has stated "Version control is indispensable on team projects. It's so effective that the applications division of Microsoft has found source code version control a major competitive advantage." (Moore, 1992) [as cited from Code Complete - A practical handbook of software construction]

Object Based Design Philosophy

This section provides an overview of the documentation provided by the CADRE Paradigm Plus software's automated OMT tool that Omnitech is currently using to define our MARS and Standardized Teleoperation System software architecture.

Although the Paradigm Plus (PP) tool's most obvious application is to facilitate software engineering, it is flexible and generic enough to allow defined objects to represent hardware, software, or a combination of both. For instance, an inertial measurement unit consists of the physical hardware as well as the data structures and routines to read the data as shown in Figures 5 and 6. .

An object might contain code and data structures that perform high level system tasks such as task planning. This module might be software only, and only interface with other software objects. objects may also contain code that is designed to be downloaded onto an EEPROM and run as lower level control or monitoring software. Then the object would include both hardware and software interfaces.

At even lower levels, other objects may represent hardware functionality such as video equipment. The designer would be able to have the hardware object contain only documentation describing the hardware and how it interfaces with the system, or the object could contain code that simulates hardware behavior.

Each of these modules encapsulates data and methods (functions) used upon the data allowing more reliable code reuse.

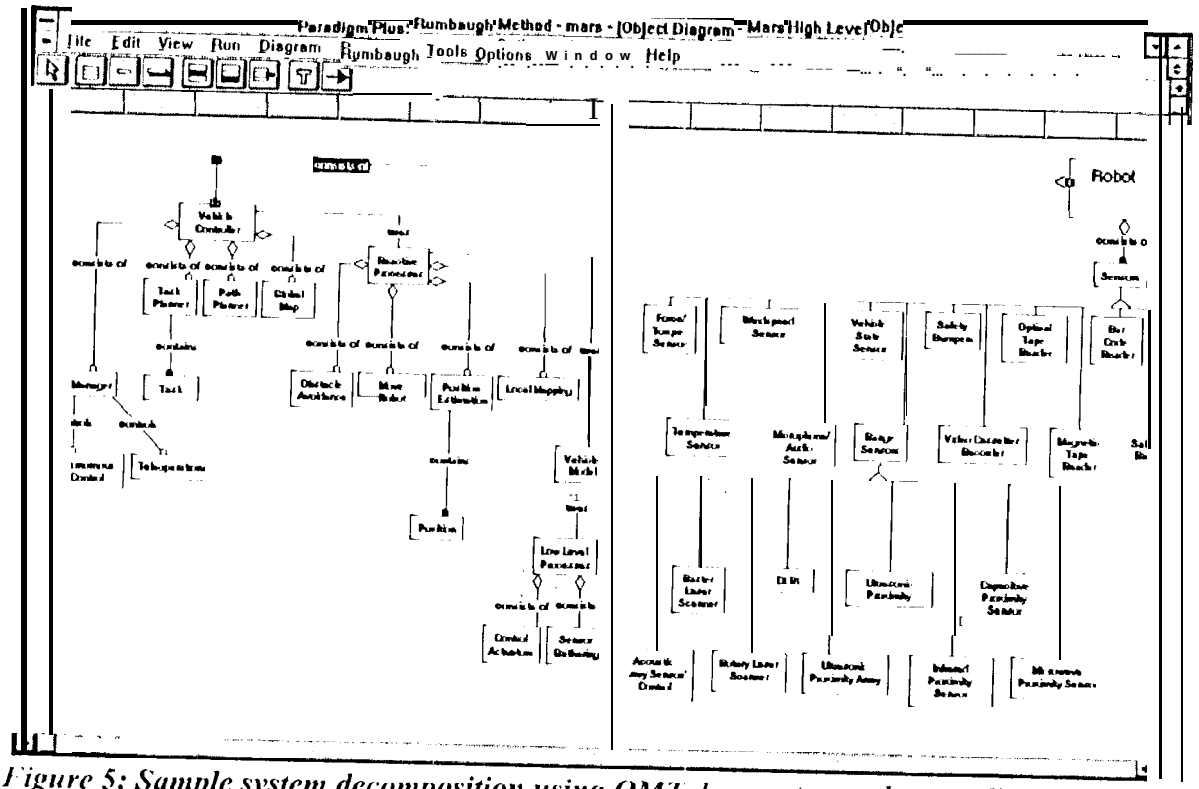

Figure 5: Sample system decomposition using OMT demonstrates the coupling of software and hardware during initial analysis.

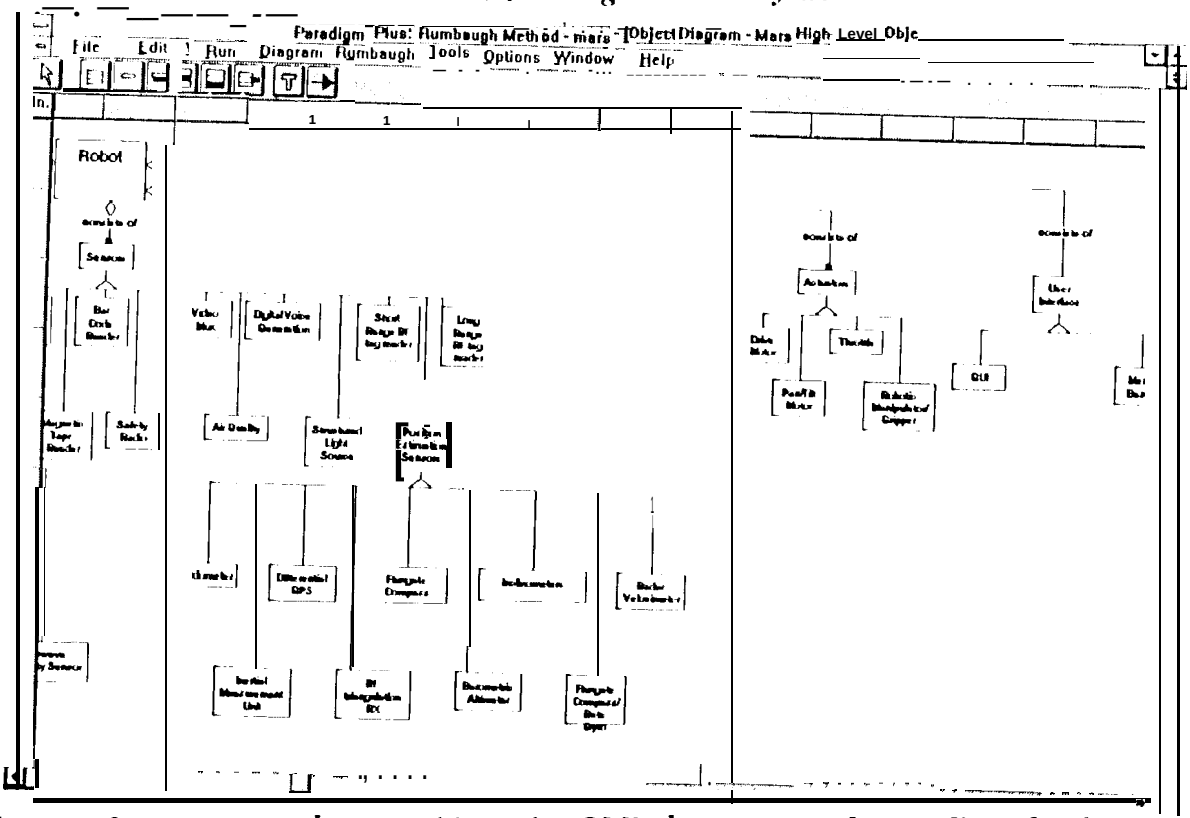

Figure 6: Sample system decomposition using OMT demonstrates the coupling of software and hardware during initial analysis.

Reusability of Code

Steve McConnel notes that "NASA's Software Engineering Laboratory studied ten projects that pursued reuse aggressively (McGarry, Waligora, and McDermott, 1989). In both the object-based and the functionally oriented approaches, the initial projects weren't able to take much of their code from previous projects because previous projects hadn't established a sufficient data base. Subsequently, the projects that used functional design were able to take about 35 percent of their code from previous projects. Projects that used an object-based approach [the approach we recommend] were able to take more than 70 percent of their code from previous projects.." [as cited from Code Complete[1] - A practical handbook of software construction]

Dynamic Modeling Documentation

One of several examples of the documentation provided by OMT, the state diagram in Figure 7 shows how events are traced through a system and provides supporting documentation for the intended use of a piece of software or hardware.

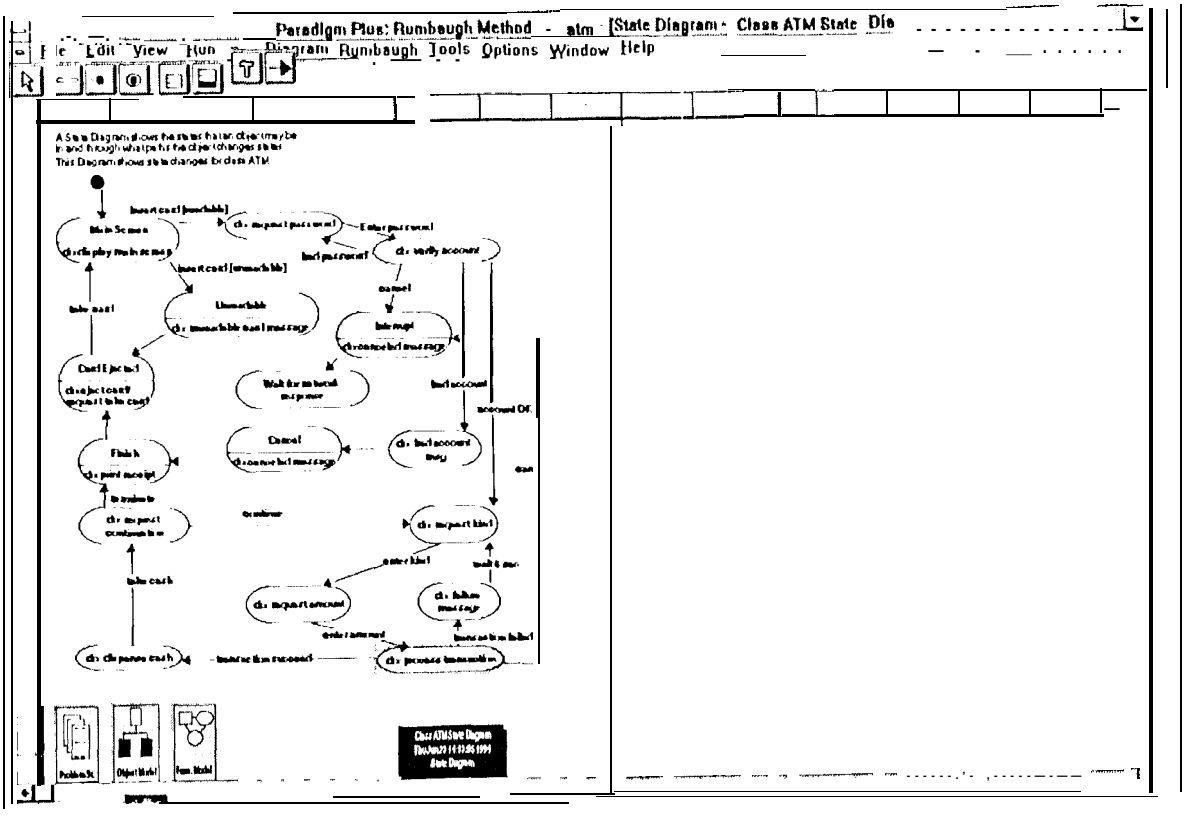

Figure 7: The dynamic modeling step in OMT provides event tracing between objects, further linking the interaction between software and hardware with the use of state diagrams.

Functional System Analysis

The functional model consists of data flow diagrams and constraints as shown in Figure 8. It identifies input and output values and defines what each function dots. Many structural analysis methods include this type of analysis, making a variety of methodologies interchangeable with the use of OMT. While a repository of software / hardware objects, such as communication protocols, would be documented and organized under OMT, this would not preclude other analytical methods from taking advantage of the modules documented with OMT.

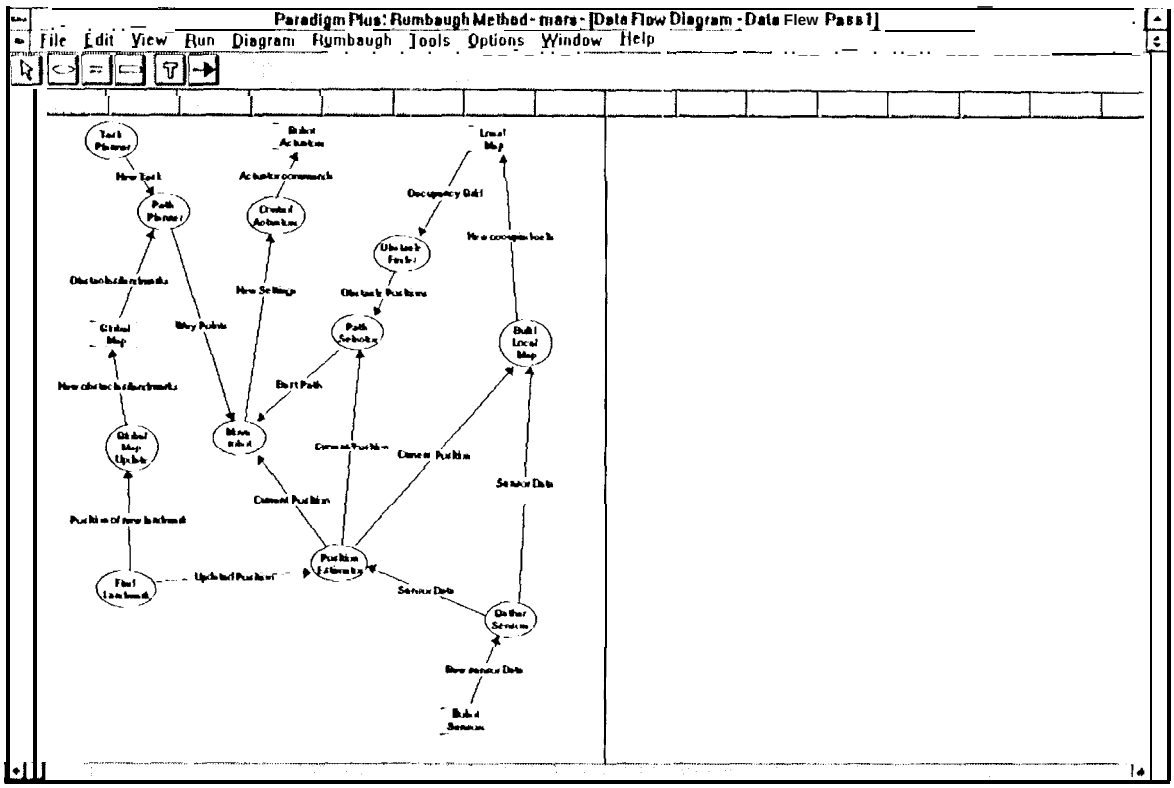

Figure 8: The OMT method then decomposes a system functionally, this is a first step in many structured analysis techniques.

Software Architecture in the OMT

Subsequent to the object modeling, dynamic modeling, and functional modeling, the overall organization of the system into subsystems comprise the system's architecture. Often the overall architecture can be based upon previous architectures. Architectures typically organize objects into a sequence of horizontal layers or and / or vertical partitions where prudent organization limits the interactions between objects, increasing modularity. An example of horizontal layering is the OSI-RM 7-layer computer communications protocol which creates definitive interfaces. An example of vertical partitioning is virtual memory management and process control in an operating system. Modularity is the key and the primary emphasis 'behind the plethora of high level architectures in the literature, variety of analysis tools, as well as classical programming practices.

The true test of modularity will be the reusability of components in future systems while maintaining some degree (preferably a high degree) of downward compatibility with present systems. The true test of a development architecture for configuration management will be its ability to control change in a fashion that does not stifle innovation.

Just as the evaluation of different objects within a system for reuse in future systems will be accomplished at the beginning of a design project by examining documentation, different high level architectures should be evaluated by reviewing intended usc statements, reproducible test cases, and a history of implementations.

Documentation Provided by Automated l'reject Management Tools

in many instances, the embedded software and hardware in a system will require that a test bed setup be provided to software developers prior to the complete system. Test beds arc sometimes necessary for a hardware system's proof of concept.

To maintain a direct correspondence between the variety of time dependent constraints imposed upon UGV systems, objects within a design can be entered directly into project management software . Maintaining project scheduling histories for components within a Configuration Management's repository would not only focus attention upon the time constraints and the cost effectiveness of modules selected for a project but also on modules under development having similar characteristics. Automated project management software provides many views of the scheduling process such as the PERT chart, overlaid calendars, and filtering of schedules to identify the usc of resources or to make calendars for different resource sharing.

Documentation and CASE tools

Our emphasis on the usc of CASE tools is stressed equally for reasons of software system configuration, and documentation to allow reuse of code. It is specifically not due to productivity gains in the original development of the code. This is less than ideal admittedly, but a reality for software development using existing CASE tools, as is evidenced in the following quote: "At the Achieving Software Quality Debates, Don Reifer reported the results of a survey on the effectiveness of CASE tools. He collected data from 45 companies in 10 industries representing over 100 million lines of code. Reifer found an average productivity gain with CASE of 9 to 12 percent, but said that not a single firm could justify the cost of CASE using the gains alone as a reason (Myers, 1992). A second report has also concluded that CASE tools have yet to meet the claims made for them in the popular press (Vessey, Jarvenpaa, and Tractinsky, 1992)." [as cited form Code Complete - A practical handbook of software construction]

l'reject Life Cycle and Cost Reduction

The life cycle of a project consists of several phases, each phase overlapping other phases to some extent. Good documentation during the design phase will contribute to the implementation phase and later to the maintenance phase. The cost of this documentation to support configuration management should not exceed its value however .